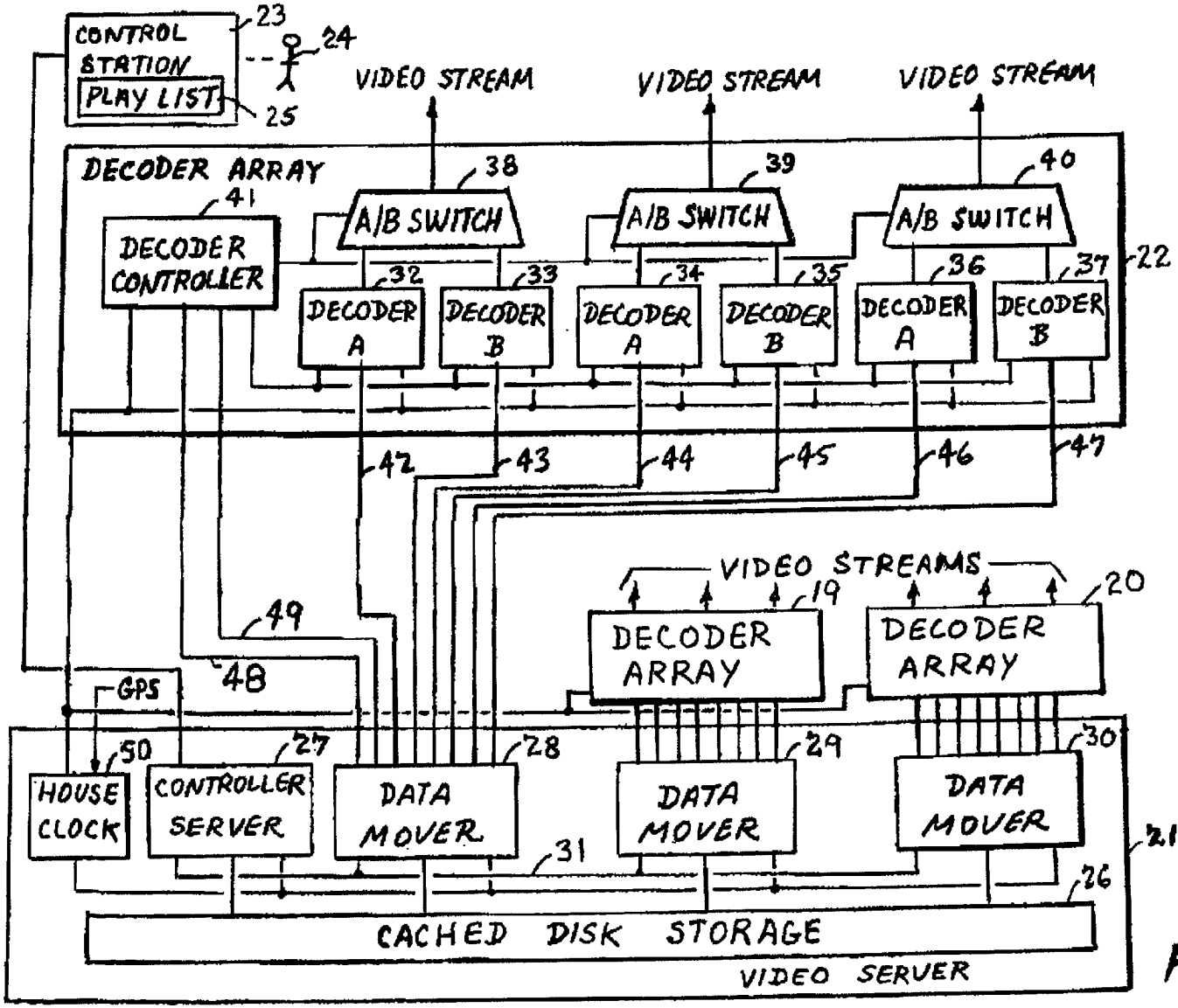
CONTROL STATION
PLAY LIST
23
24
25
VIDEO STREAM
VIDEO STREAM
VIDEO STREAM
DECODER ARRAY
41
DECODER CONTROLLER
38
A/B SWITCH
39
A/B SWITCH
40
A/B SWITCH
32
DECODER A
33
DECODER B
34
DECODER A
35
DECODER B
36
DECODER A
37
DECODER B
22
42
43
44
45
46
47
VIDEO STREAMS
19
20
DECODER ARRAY
DECODER ARRAY
49
48
GPS
50
27
28
29
30
HOUSE CLOCK
CONTROLLER SERVER
DATA MOVER
DATA MOVER
DATA MOVER
31
21
26
CACHED DISK STORAGE
VIDEO SERVER

FIG. 1

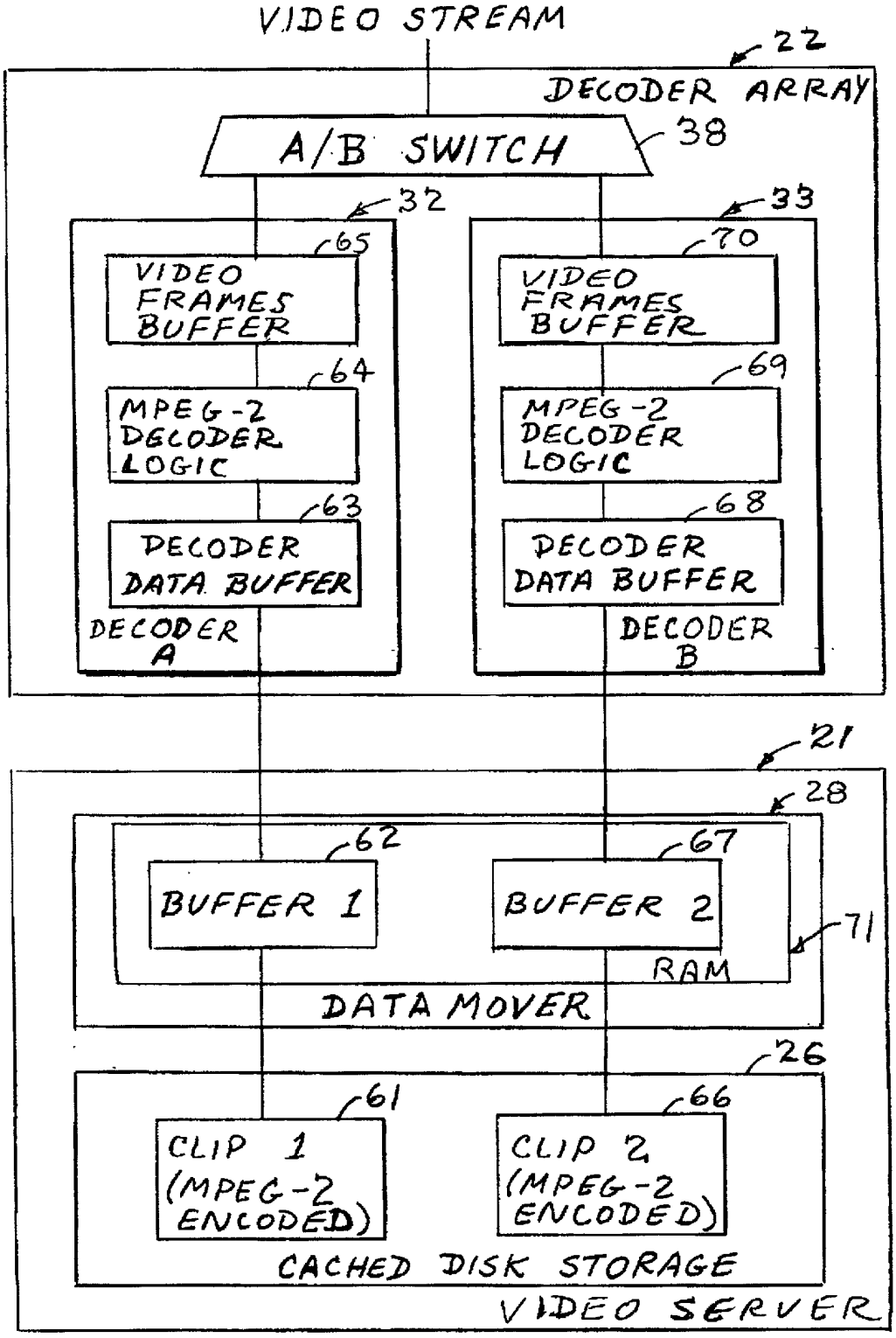
VIDEO STREAM
22
DECODER ARRAY
A/B SWITCH
38
32
33
65
70
VIDEO FRAMES BUFFER
VIDEO FRAMES BUFFER
64
69
MPEG-2 DECODER LOGIC
MPEG-2 DECODER LOGIC
63
68
DECODER DATA BUFFER
DECODER DATA BUFFER
DECODER A
DECODER B
21
28
62
67
BUFFER 1
BUFFER 2
71
RAM
DATA MOVER
26
61
66
CLIP 1 (MPEG-2 ENCODED)
CLIP 2 (MPEG-2 ENCODED)
CACHED DISK STORAGE
VIDEO SERVER

FIG. 2

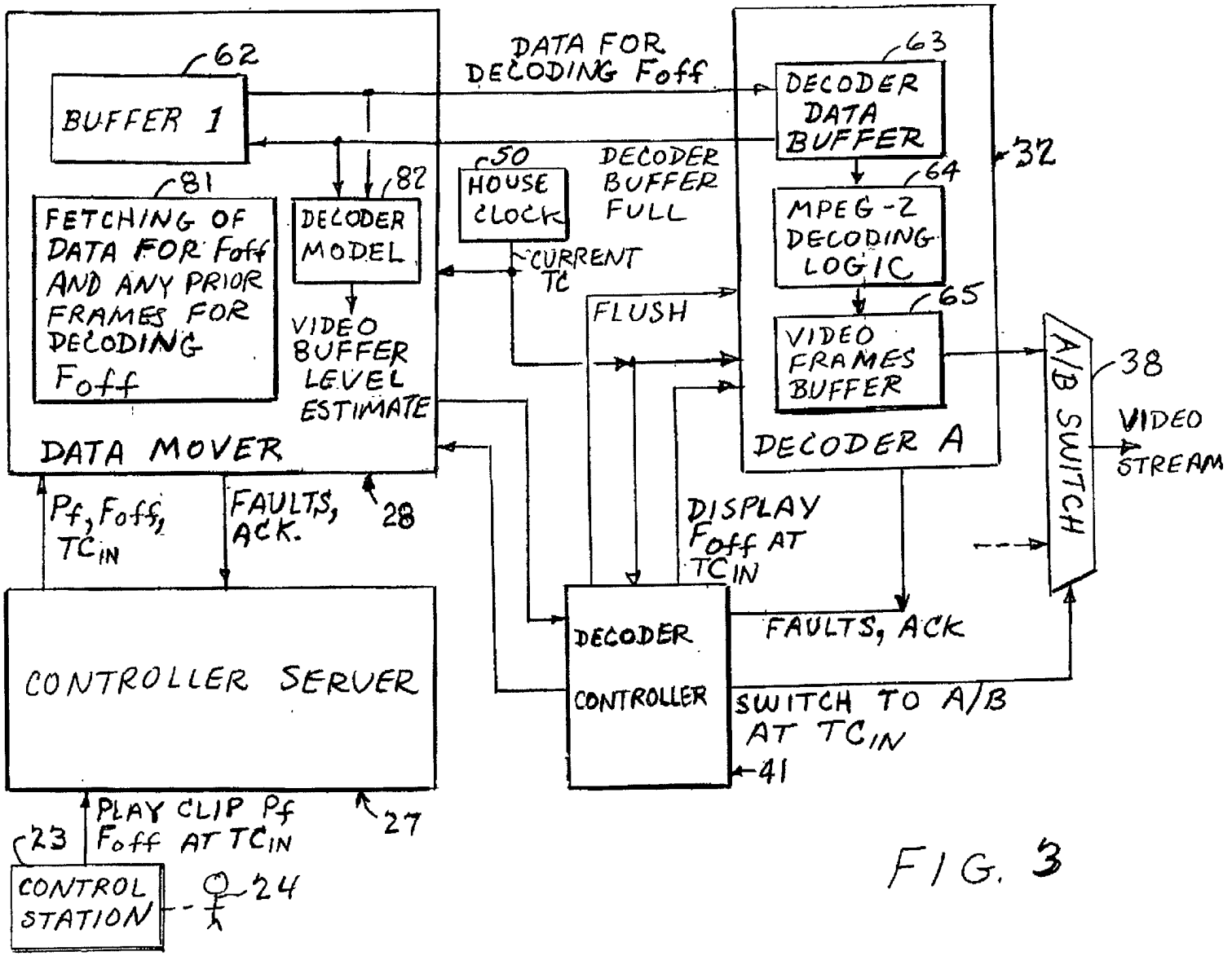


FIG. 3

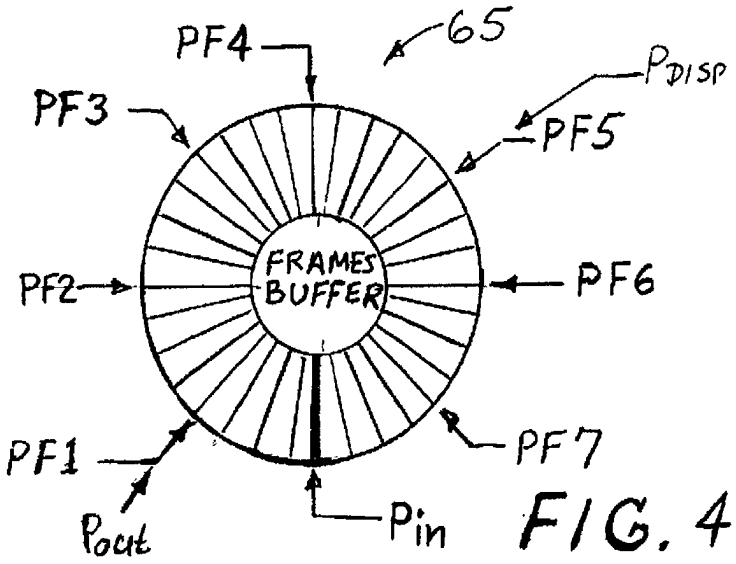


FIG. 4

| | | | |
|---|---|---|---|
| DISPLAY | $P_{DISP}$ | | |
| IN | $P_{IN}$ | | |
| OUT | $P_{OUT}$ | | |
| NO. FRAMES | 7 | | |
| F1 | FOFF1 | TC1 | PF1 |
| F2 | FOFF2 | TC2 | PF2 |
| F3 | FOFF3 | TC3 | PF3 |
| F4 | FOFF4 | TC4 | PF4 |
| F5 | FOFF5 | TC5 | PF5 |
| F6 | FOFF6 | TC6 | PF6 |
| F7 | FOFF7 | TC7 | PF7 |

FIG. 5

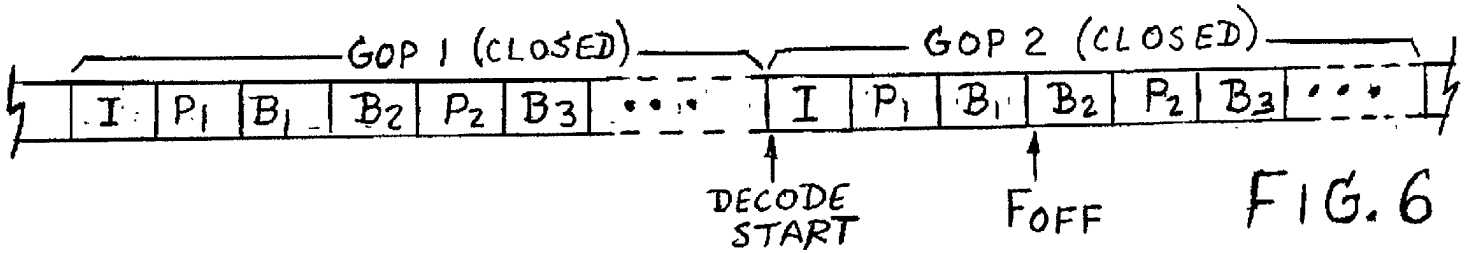


FIG. 6

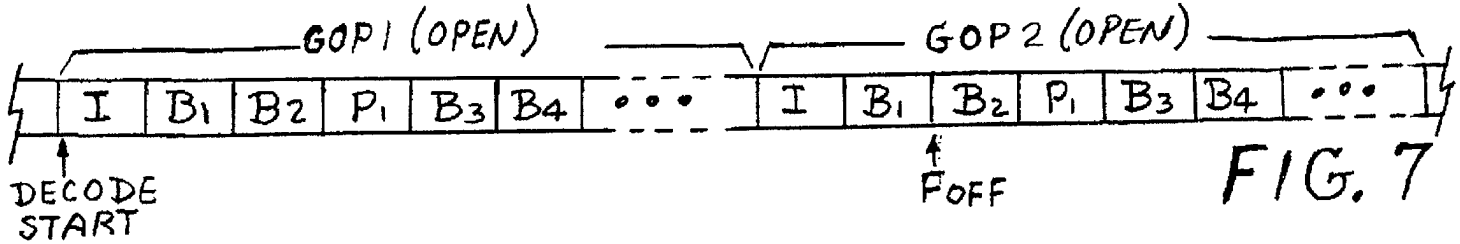


FIG. 7

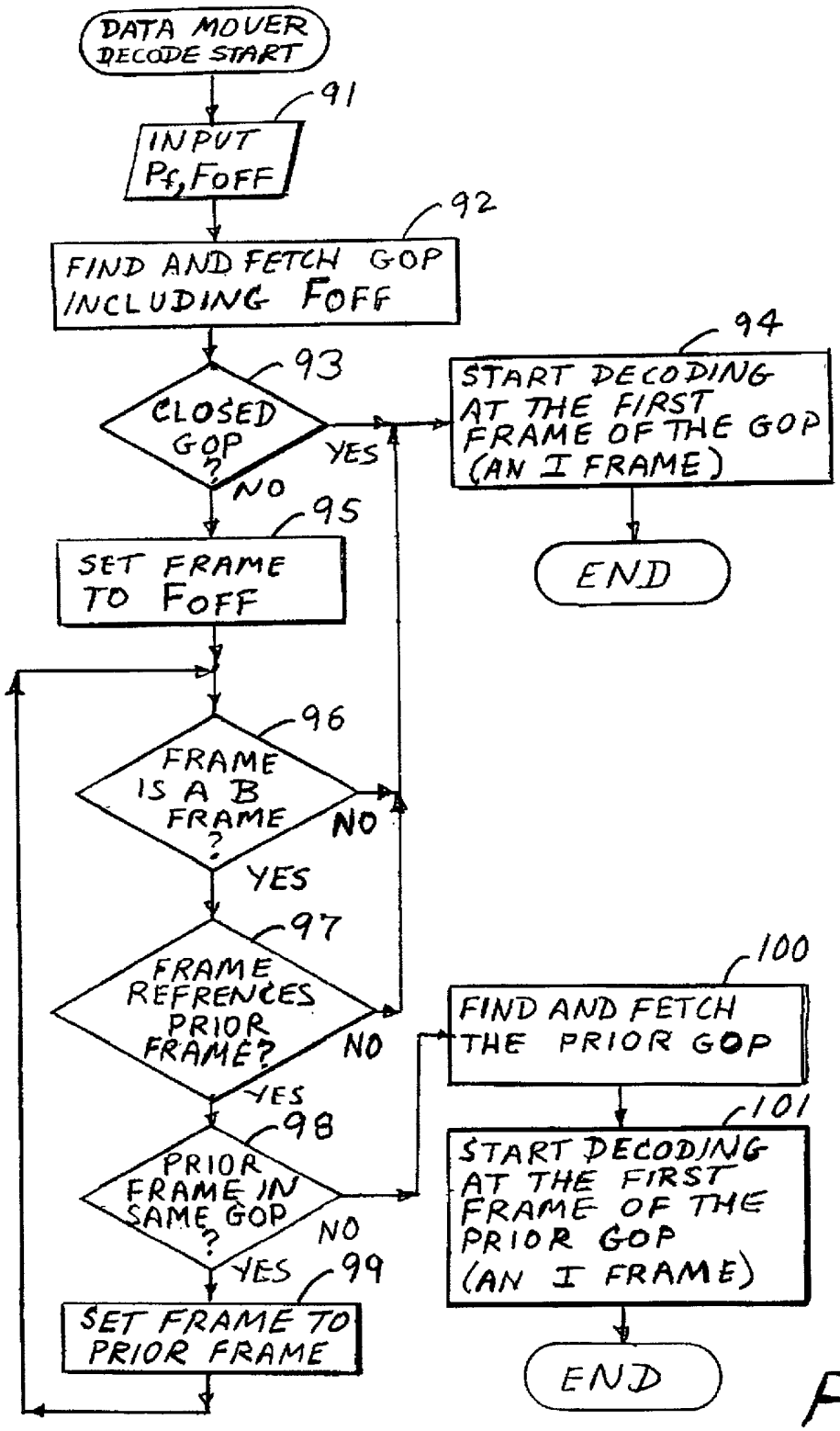
DATA MOVER DECODE START
91
INPUT Pf, FOFF
92
FIND AND FETCH GOP INCLUDING FOFF
93
CLOSED GOP ?
YES
NO
94
START DECODING AT THE FIRST FRAME OF THE GOP (AN I FRAME)
END
95
SET FRAME TO FOFF
96
FRAME IS A B FRAME ?
NO
YES
97
FRAME REFRENCES PRIOR FRAME?
NO
YES
98
PRIOR FRAME IN SAME GOP ?
NO
YES
99
SET FRAME TO PRIOR FRAME
100
FIND AND FETCH THE PRIOR GOP
101
START DECODING AT THE FIRST FRAME OF THE PRIOR GOP (AN I FRAME)
END

FIG. 8

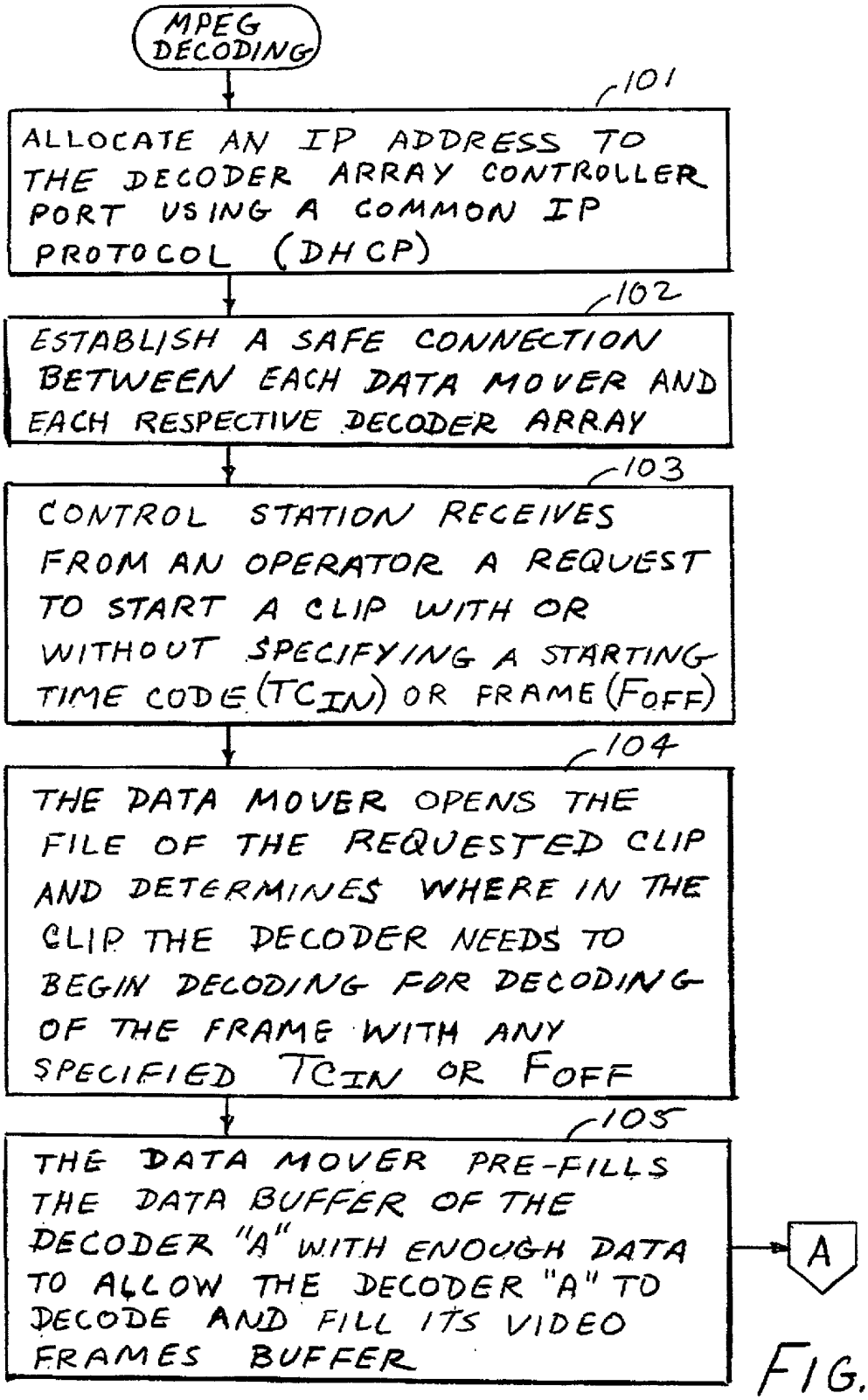
MPEG DECODING
101
ALLOCATE AN IP ADDRESS TO THE DECODER ARRAY CONTROLLER PORT USING A COMMON IP PROTOCOL (DHCP)
102
ESTABLISH A SAFE CONNECTION BETWEEN EACH DATA MOVER AND EACH RESPECTIVE DECODER ARRAY
103
CONTROL STATION RECEIVES FROM AN OPERATOR A REQUEST TO START A CLIP WITH OR WITHOUT SPECIFYING A STARTING TIME CODE ($TC_{IN}$) OR FRAME ($F_{OFF}$)
104
THE DATA MOVER OPENS THE FILE OF THE REQUESTED CLIP AND DETERMINES WHERE IN THE CLIP THE DECODER NEEDS TO BEGIN DECODING FOR DECODING OF THE FRAME WITH ANY SPECIFIED $TC_{IN}$ OR $F_{OFF}$
105
THE DATA MOVER PRE-FILLS THE DATA BUFFER OF THE DECODER "A" WITH ENOUGH DATA TO ALLOW THE DECODER "A" TO DECODE AND FILL ITS VIDEO FRAMES BUFFER
A

FIG. 9

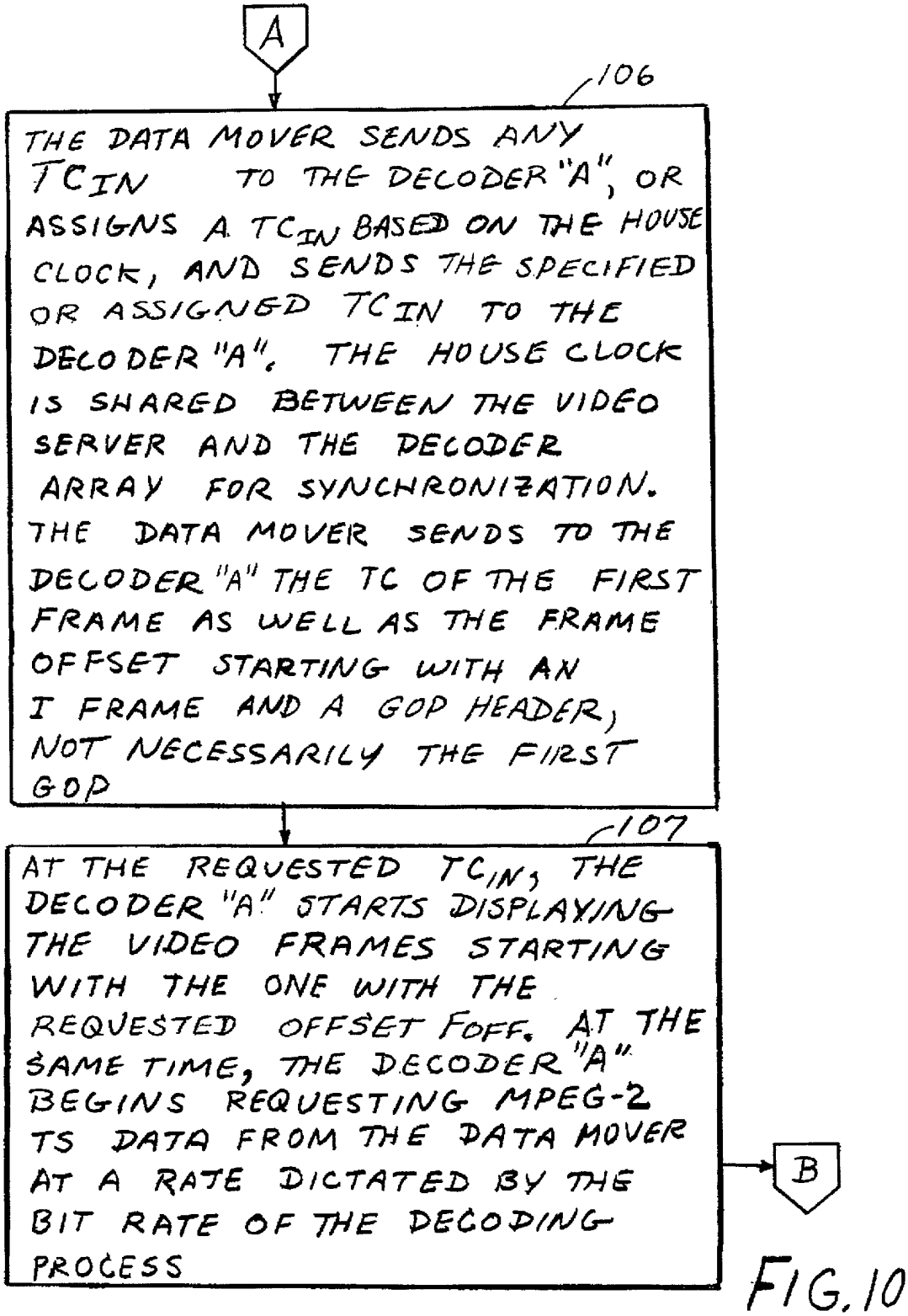
A
106
THE DATA MOVER SENDS ANY $TC_{IN}$ TO THE DECODER "A", OR ASSIGNS A $TC_{IN}$ BASED ON THE HOUSE CLOCK, AND SENDS THE SPECIFIED OR ASSIGNED $TC_{IN}$ TO THE DECODER "A". THE HOUSE CLOCK IS SHARED BETWEEN THE VIDEO SERVER AND THE DECODER ARRAY FOR SYNCHRONIZATION. THE DATA MOVER SENDS TO THE DECODER "A" THE TC OF THE FIRST FRAME AS WELL AS THE FRAME OFFSET STARTING WITH AN I FRAME AND A GOP HEADER, NOT NECESSARILY THE FIRST GOP
107
AT THE REQUESTED $TC_{IN}$, THE DECODER "A" STARTS DISPLAYING THE VIDEO FRAMES STARTING WITH THE ONE WITH THE REQUESTED OFFSET $F_{OFF}$. AT THE SAME TIME, THE DECODER "A" BEGINS REQUESTING MPEG-2 TS DATA FROM THE DATA MOVER AT A RATE DICTATED BY THE BIT RATE OF THE DECODING PROCESS
B

FIG. 10

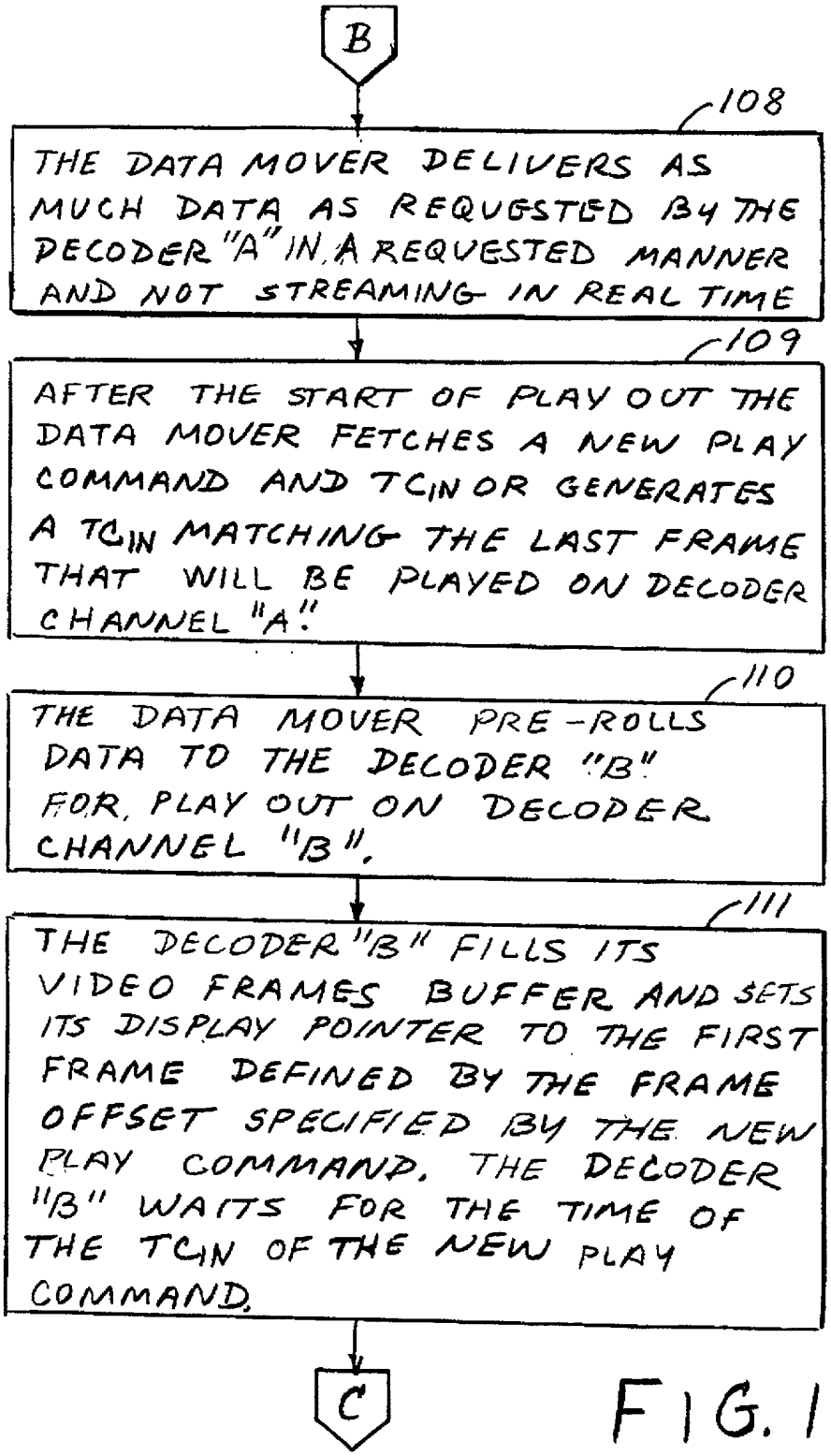
B
108
THE DATA MOVER DELIVERS AS MUCH DATA AS REQUESTED BY THE DECODER "A" IN A REQUESTED MANNER AND NOT STREAMING IN REAL TIME
109
AFTER THE START OF PLAY OUT THE DATA MOVER FETCHES A NEW PLAY COMMAND AND TCIN OR GENERATES A TCIN MATCHING THE LAST FRAME THAT WILL BE PLAYED ON DECODER CHANNEL "A".
110
THE DATA MOVER PRE-ROLLS DATA TO THE DECODER "B" FOR PLAY OUT ON DECODER CHANNEL "B".
111
THE DECODER "B" FILLS ITS VIDEO FRAMES BUFFER AND SETS ITS DISPLAY POINTER TO THE FIRST FRAME DEFINED BY THE FRAME OFFSET SPECIFIED BY THE NEW PLAY COMMAND. THE DECODER "B" WAITS FOR THE TIME OF THE TCIN OF THE NEW PLAY COMMAND.
C

FIG. 11

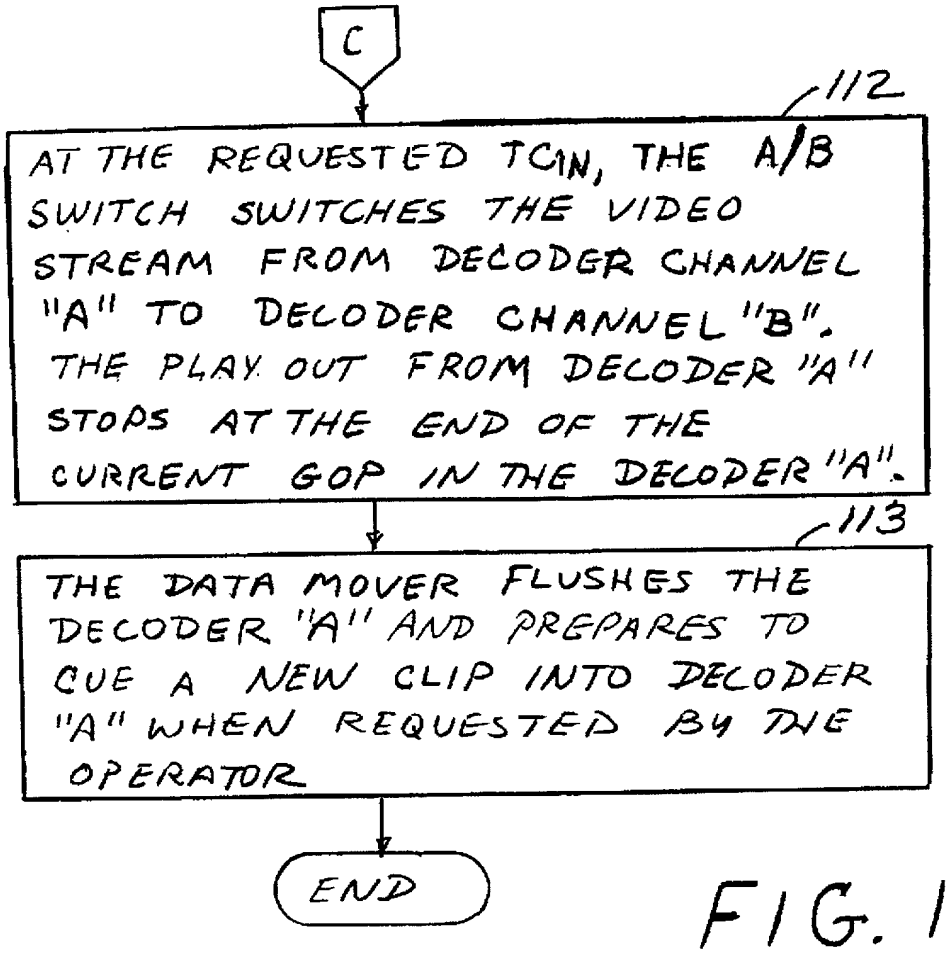
C
112
AT THE REQUESTED TCIN, THE A/B SWITCH SWITCHES THE VIDEO STREAM FROM DECODER CHANNEL "A" TO DECODER CHANNEL "B". THE PLAY OUT FROM DECODER "A" STOPS AT THE END OF THE CURRENT GOP IN THE DECODER "A".
113
THE DATA MOVER FLUSHES THE DECODER "A" AND PREPARES TO CUE A NEW CLIP INTO DECODER "A" WHEN REQUESTED BY THE OPERATOR
END

FIG. 12

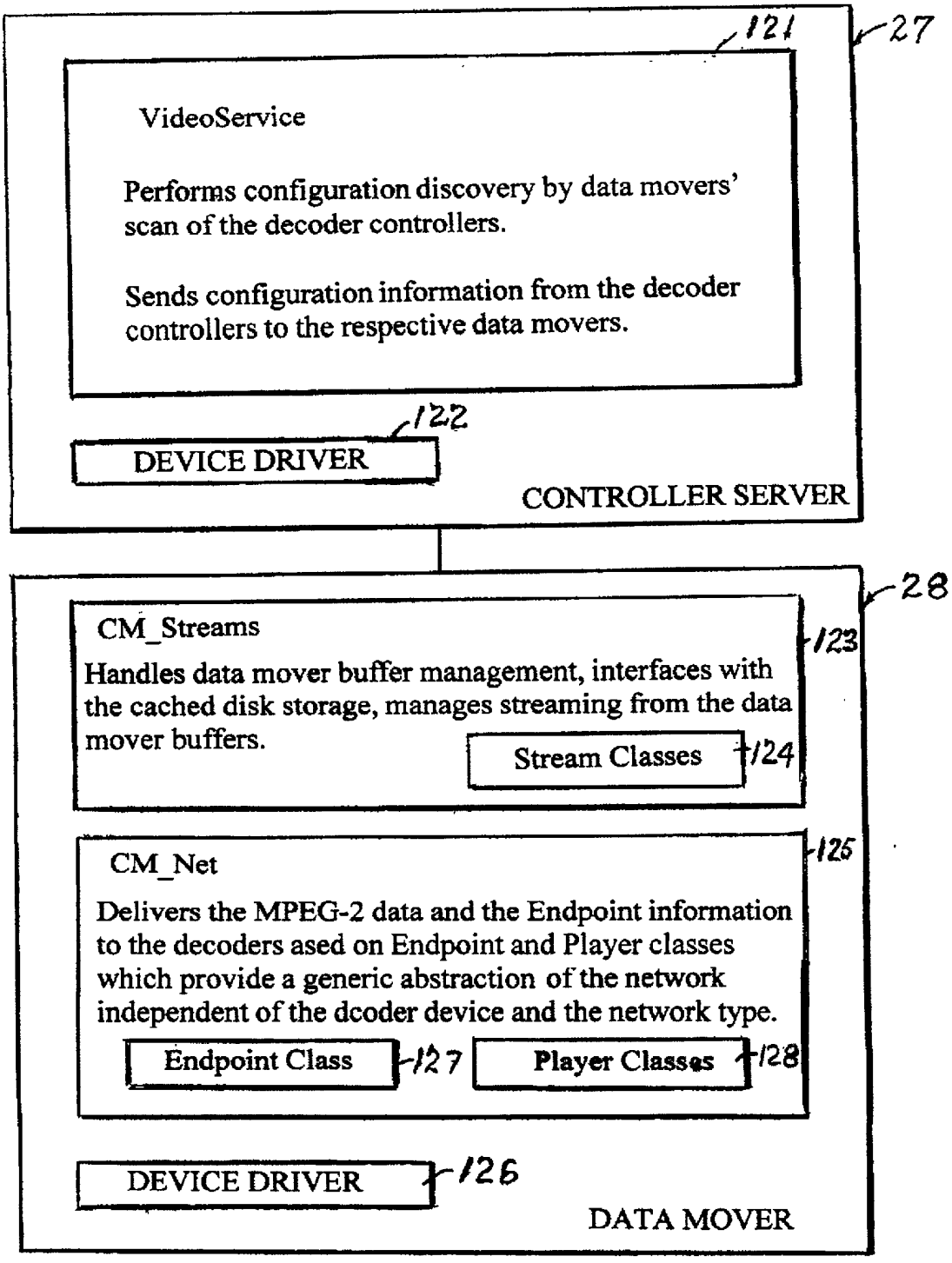
121
27
VideoService
Performs configuration discovery by data movers' scan of the decoder controllers.
Sends configuration information from the decoder controllers to the respective data movers.
122
DEVICE DRIVER
CONTROLLER SERVER
28
CM_Streams
123
Handles data mover buffer management, interfaces with the cached disk storage, manages streaming from the data mover buffers.
Stream Classes
124
125
CM_Net
Delivers the MPEG-2 data and the Endpoint information to the decoders ased on Endpoint and Player classes which provide a generic abstraction of the network independent of the dcoder device and the network type.
Endpoint Class
127
Player Classes
128
DEVICE DRIVER
126
DATA MOVER

FIG. 13

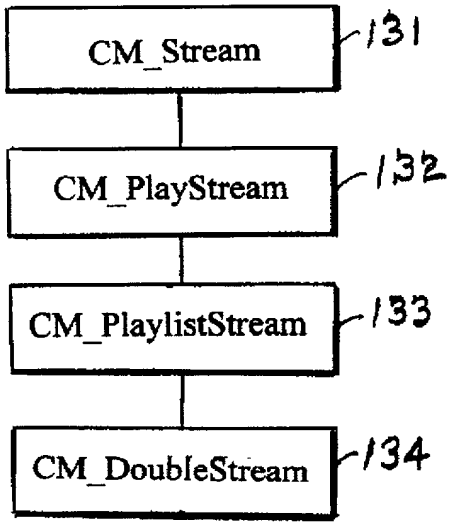
CM_Stream
131
CM_PlayStream
132
CM_PlaylistStream
133
CM_DoubleStream
134

FIG. 14

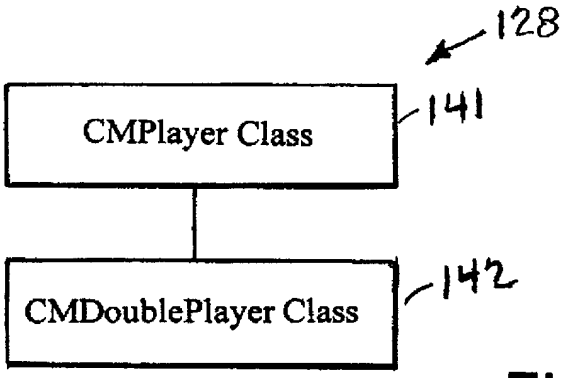
128
CMPlayer Class
141
CMDoublePlayer Class
142

FIG. 15

**Control Protocol**

| |
|---|
| Configuration.<br>Allows the data mover to determine the configuration of the decoder array and set up any configuration parameters.<br>Commands: QueryStatus, Configure |
| Streaming.<br>Controls delivery of streams (i.e., timing, clips, transition type).<br>Commands: PrerollClip, ModifyDisplayTime, CancelClipPreroll, PauseClip, ResumeClip, ModifyClip. |
| Asynchronous status reports.<br>Asynchronous reports of significant events from the decoder array to the data mover.<br>Commands: ClipHasStarted, ClipHasEnded, ClipIsEnding, TrapMessage, EditSummary, |
| Edit.<br>Allows all decoders in the decoder array to be controlled by an edit review station.<br>Commmands: Jog forward/backward, Shuttle forward/backward, Stop, Goto a specific timecode, and normal Play. |

FIG. 16

**Format of Streaming Protocol Ethernet Packet**

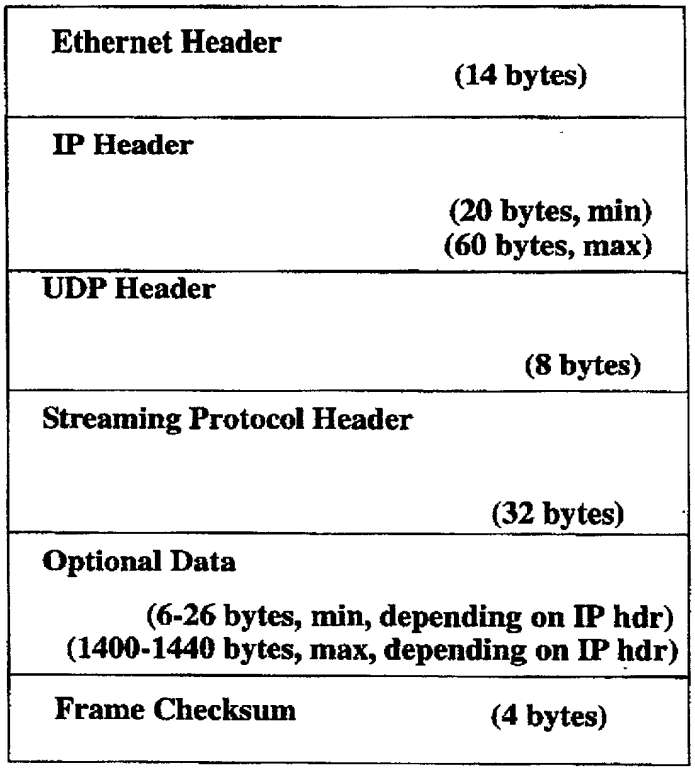


FIG. 17

**Request Message Header Format**

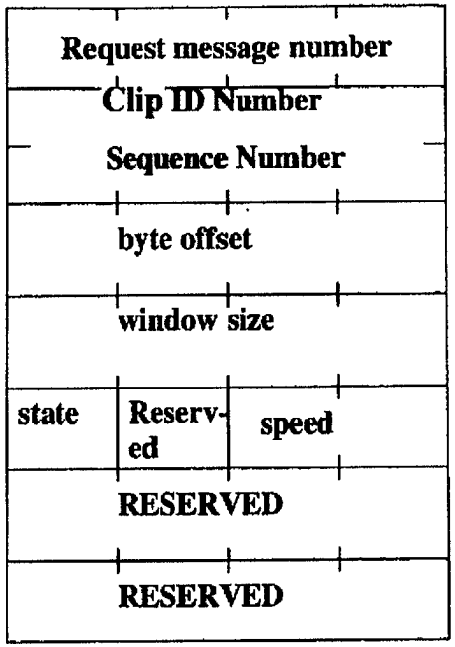


FIG. 18

**Data Message Header Format**

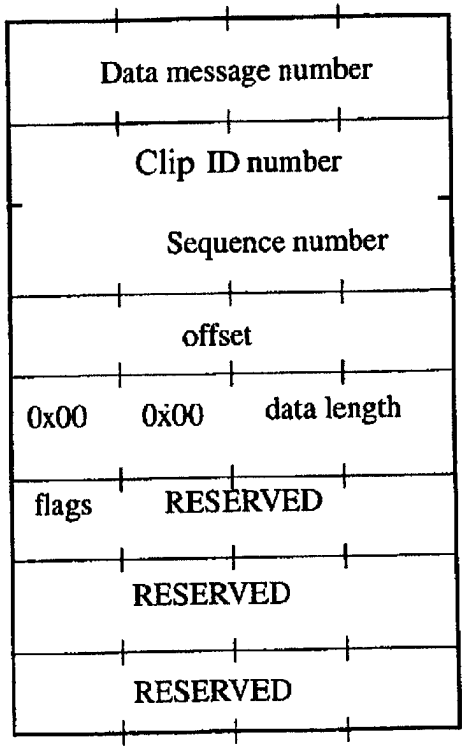


FIG. 19

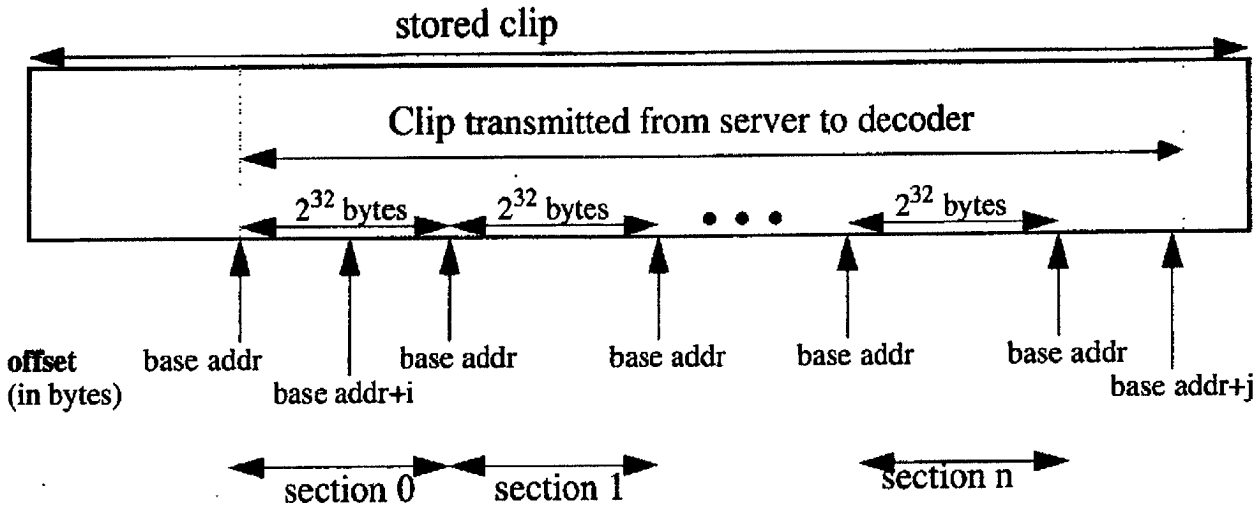
stored clip
Clip transmitted from server to decoder
$2^{32}$ bytes
$2^{32}$ bytes
$2^{32}$ bytes
offset
(in bytes)
base addr
base addr+i
base addr
base addr
base addr
base addr
base addr+j
section 0
section 1
section n

FIG. 20

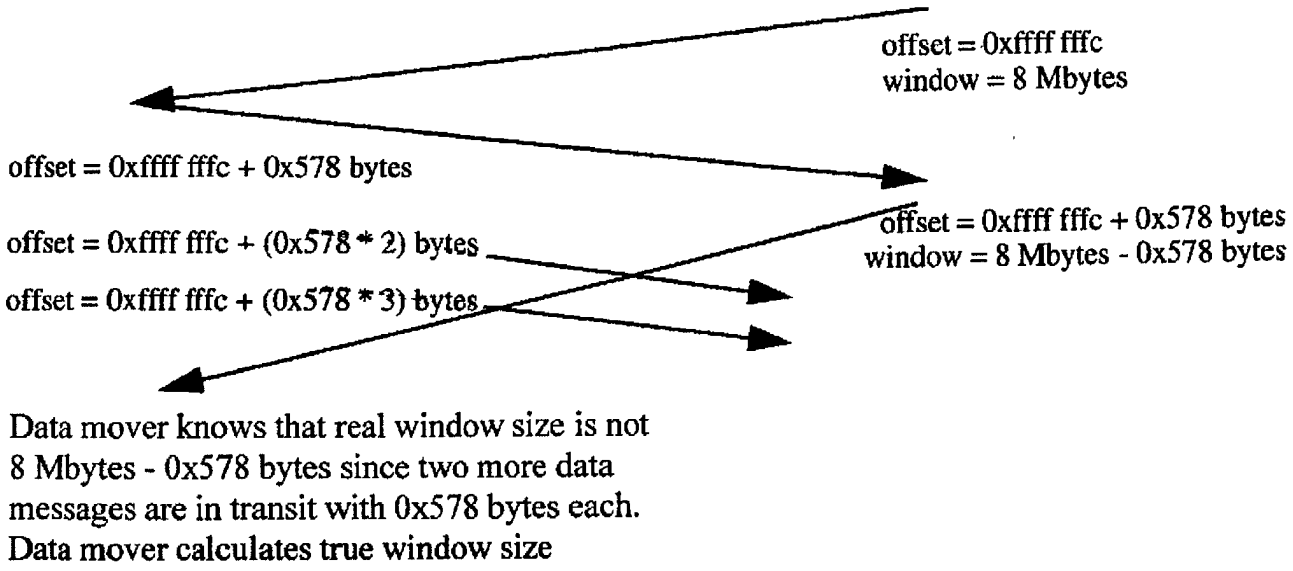
Data Mover sets offset in Data Message
Decoder sets offset in Receive Message
offset = 0xffff fffc
window = 8 Mbytes
offset = 0xffff fffc + 0x578 bytes
offset = 0xffff fffc + (0x578 * 2) bytes
offset = 0xffff fffc + (0x578 * 3) bytes
offset = 0xffff fffc + 0x578 bytes
window = 8 Mbytes - 0x578 bytes
Data mover knows that real window size is not
8 Mbytes - 0x578 bytes since two more data
messages are in transit with 0x578 bytes each.
Data mover calculates true window size

FIG. 21

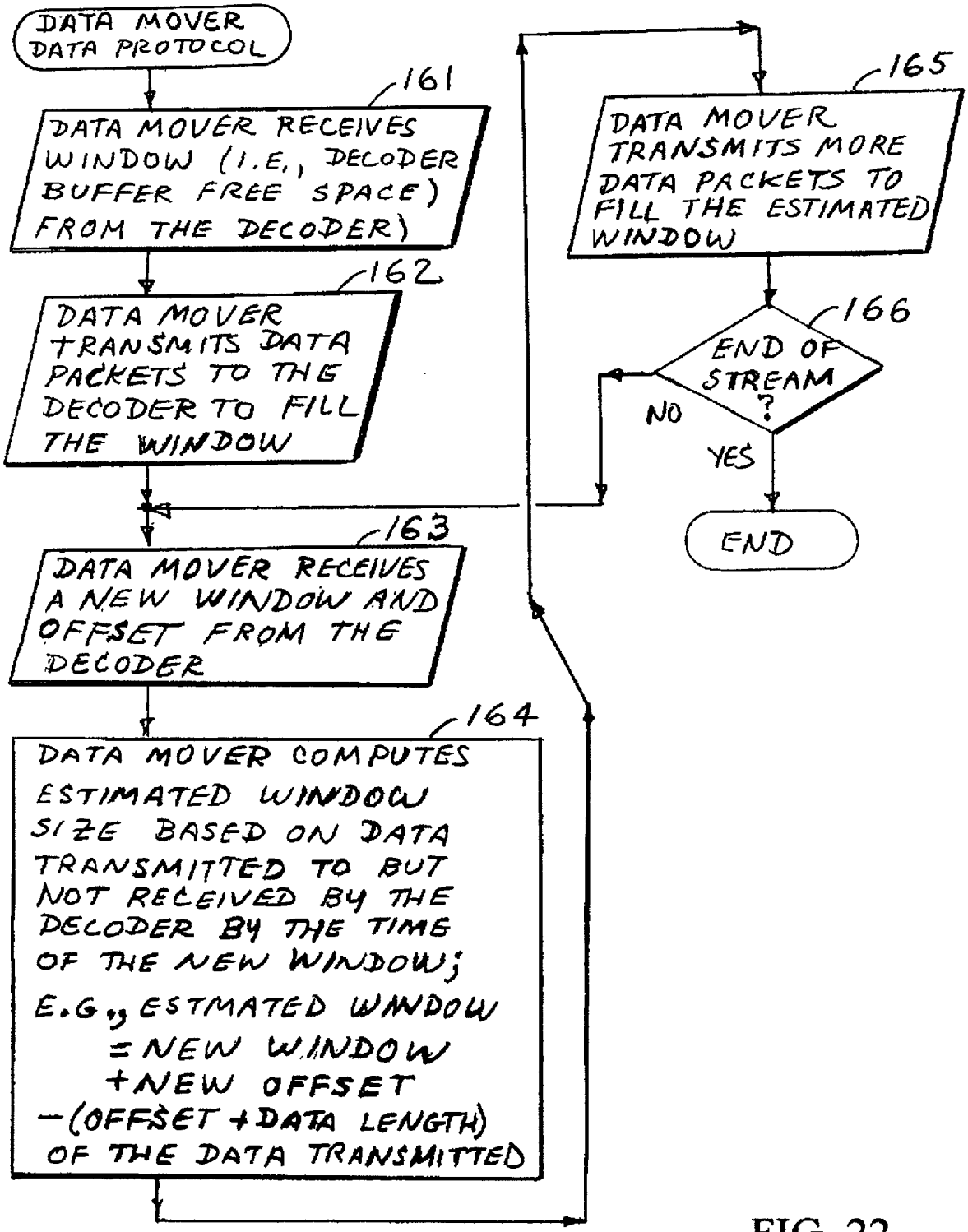
DATA MOVER DATA PROTOCOL
161
DATA MOVER RECEIVES WINDOW (I.E., DECODER BUFFER FREE SPACE) FROM THE DECODER)
162
DATA MOVER TRANSMITS DATA PACKETS TO THE DECODER TO FILL THE WINDOW
163
DATA MOVER RECEIVES A NEW WINDOW AND OFFSET FROM THE DECODER
164
DATA MOVER COMPUTES ESTIMATED WINDOW SIZE BASED ON DATA TRANSMITTED TO BUT NOT RECEIVED BY THE DECODER BY THE TIME OF THE NEW WINDOW; E.G., ESTMATED WNDOW = NEW WINDOW + NEW OFFSET -(OFFSET + DATA LENGTH) OF THE DATA TRANSMITTED
165
DATA MOVER TRANSMITS MORE DATA PACKETS TO FILL THE ESTIMATED WINDOW
166
END OF STREAM ?
NO
YES
END

FIG. 22

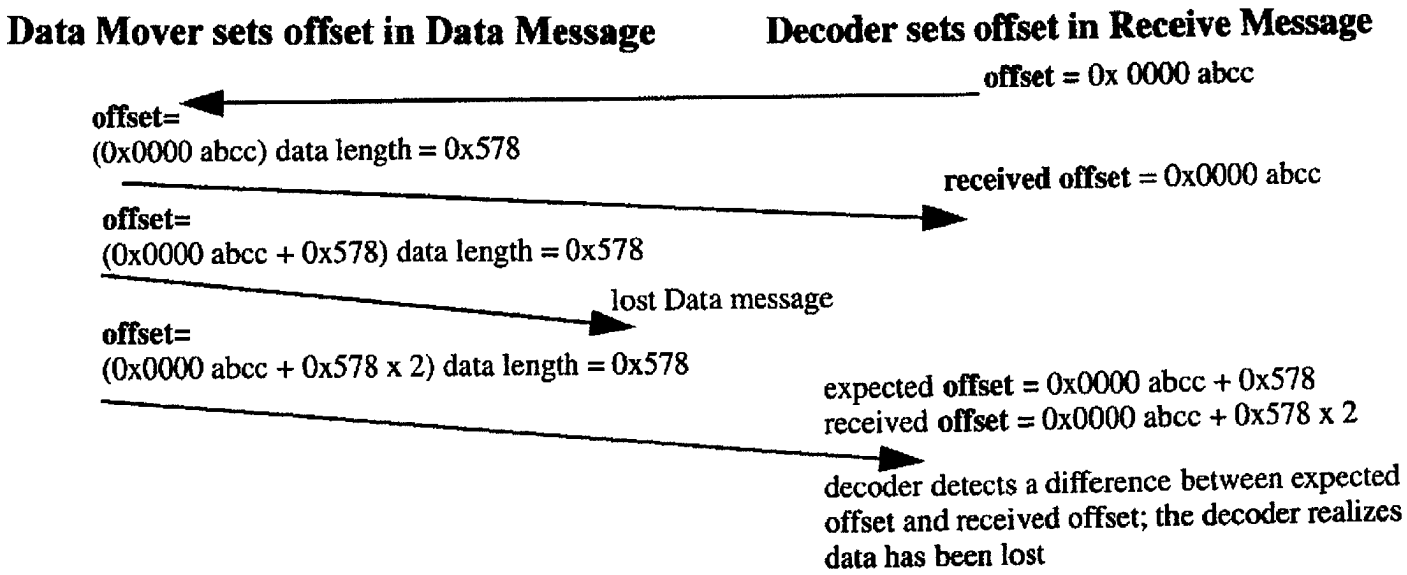
Data Mover sets offset in Data Message
Decoder sets offset in Receive Message
offset = 0x 0000 abcc
offset=
(0x0000 abcc) data length = 0x578
received offset = 0x0000 abcc
offset=
(0x0000 abcc + 0x578) data length = 0x578
lost Data message
offset=
(0x0000 abcc + 0x578 x 2) data length = 0x578
expected offset = 0x0000 abcc + 0x578
received offset = 0x0000 abcc + 0x578 x 2
decoder detects a difference between expected offset and received offset; the decoder realizes data has been lost

FIG. 23

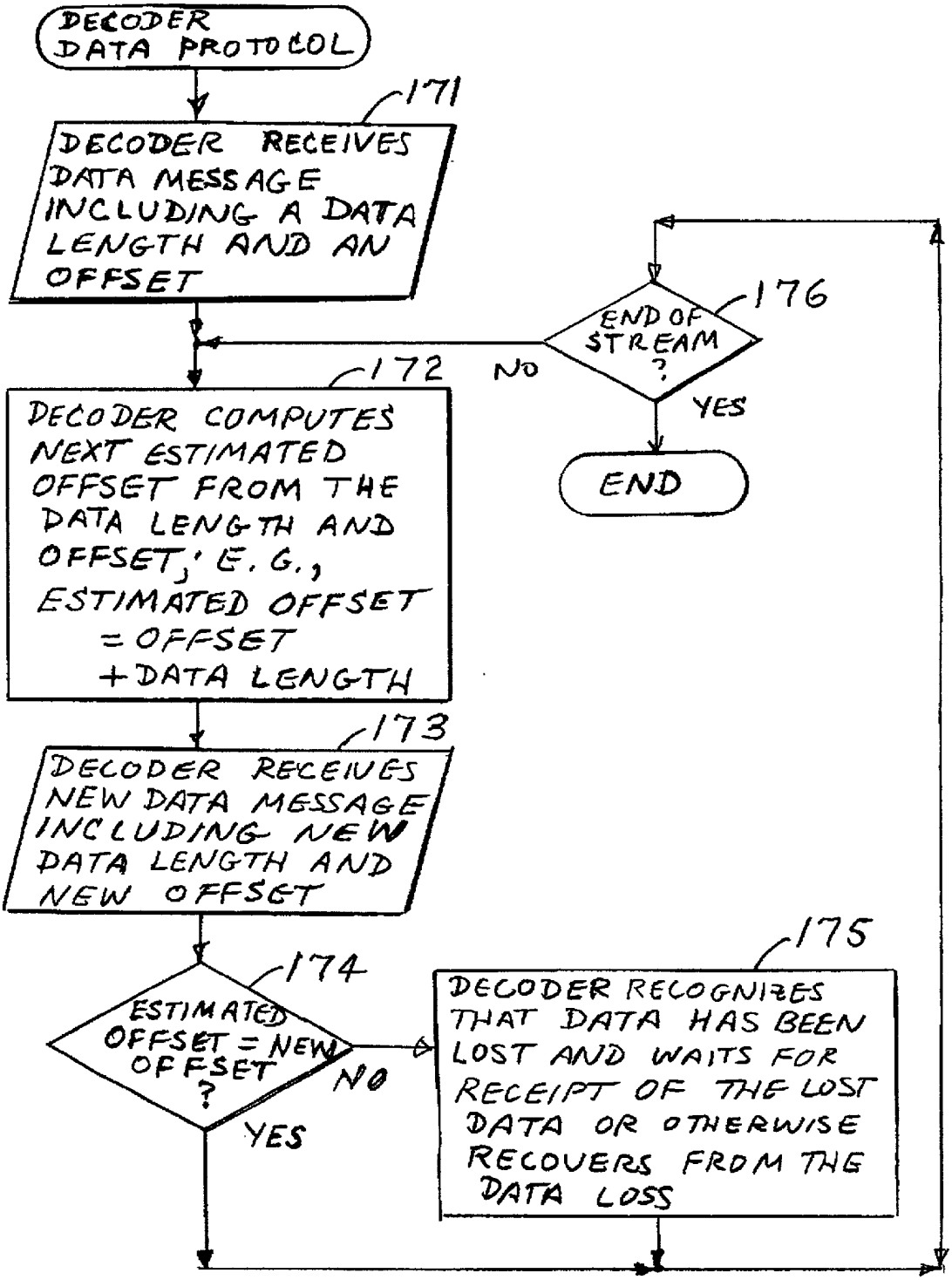
DECODER DATA PROTOCOL
171
DECODER RECEIVES DATA MESSAGE INCLUDING A DATA LENGTH AND AN OFFSET
176
END OF STREAM ?
NO
YES
END
172
DECODER COMPUTES NEXT ESTIMATED OFFSET FROM THE DATA LENGTH AND OFFSET, E.G., ESTIMATED OFFSET = OFFSET + DATA LENGTH
173
DECODER RECEIVES NEW DATA MESSAGE INCLUDING NEW DATA LENGTH AND NEW OFFSET
174
ESTIMATED OFFSET = NEW OFFSET ?
NO
YES
175
DECODER RECOGNIZES THAT DATA HAS BEEN LOST AND WAITS FOR RECEIPT OF THE LOST DATA OR OTHERWISE RECOVERS FROM THE DATA LOSS

FIG. 24

**Definition of Streaming States**

| Streaming State | Definition |
|---|---|
| Cueing | Data mover sends data to the decoder, at least up to the time code that must be displayed. The data rate can be at a rate convenient for the data mover. The decoder consumes the data at 1x real time. It is not important if the decoder underflows, since the underflow would be before the display time. |
| Streaming | The data mover sends data to the decoder at 1x real time and the decoder consumes the data at 1x real time; the decoder can under-flow/overflow and it will affect the picture presented to the viewer. |
| Stopped | The decoder is not consuming data. During this state, the decoder continues to send Request messages at the configured Request interval. |
| non-overlapped | This state requires that the decoder send a new Request message only after receiving a response from the previous Request message. The data mover may use this mode for testing. |

FIG. 25

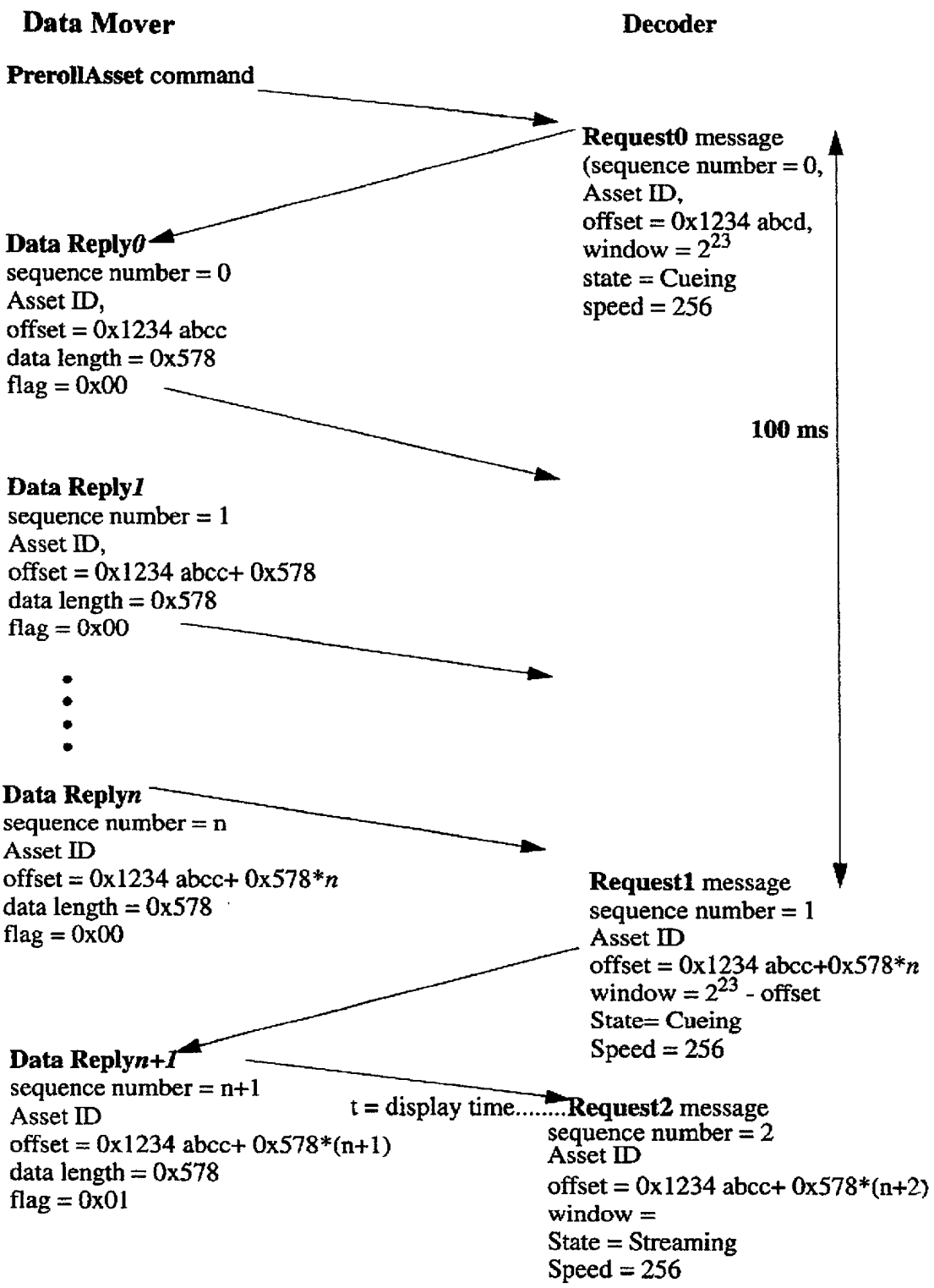
Data Mover
Decoder
PrerollAsset command
Request0 message
(sequence number = 0,
Asset ID,
offset = 0x1234 abcd,
window = 2^23
state = Cueing
speed = 256
Data Reply0
sequence number = 0
Asset ID,
offset = 0x1234 abcc
data length = 0x578
flag = 0x00
100 ms
Data Reply1
sequence number = 1
Asset ID,
offset = 0x1234 abcc+ 0x578
data length = 0x578
flag = 0x00
Data Replyn
sequence number = n
Asset ID
offset = 0x1234 abcc+ 0x578*n
data length = 0x578
flag = 0x00
Request1 message
sequence number = 1
Asset ID
offset = 0x1234 abcc+0x578*n
window = 2^23 - offset
State= Cueing
Speed = 256
Data Replyn+1
sequence number = n+1
Asset ID
offset = 0x1234 abcc+ 0x578*(n+1)
data length = 0x578
flag = 0x01
t = display time.....
Request2 message
sequence number = 2
Asset ID
offset = 0x1234 abcc+ 0x578*(n+2)
window =
State = Streaming
Speed = 256

FIG. 26